Resilient Data Aggregation for Unattended WSNs

Jens-Matthias Bohli*, Panos Papadimitratos†, Donato Verardi‡ and Dirk Westhoff§
*NEC Laboratories Europe, Kurfürsten-Anlage 36, 69115 Heidelberg, Germany
Email: bohli@neclab.eu
†School of Electrical Engineering, KTH Stockholm, Sweden
Email: papadim@kth.se, www.ee.kth.se/~papadim/
‡Trivadis Holding AG, Switzerland
§Department of Computer Science, HAW Hamburg, Berliner Tor 7, 20099 Hamburg, Germany
Email: westhoff@informatik.haw-hamburg.de

***Abstract*—Unattended wireless sensor networks (WSNs) collect and store sensed data in the absence of a base station (sink). WSN data aggregation is a widely accepted approach to improve storage and communication efficiency. But the vulnerability of low-cost WSN nodes to compromise makes the use of secure protocols mandatory. As most secure data aggregation algorithms use the base station as a trust anchor, unattended WSNs need new solutions for secure data aggregation. We address exactly this problem, proposing a new resilient data aggregation scheme that protects data *integrity* and remains *robust* to a wide range of attacks, integrating *Quality-of-Information* (QoI) as a defense mechanism. We argue that a QoI metric accompanying every aggregation result is necessary for the WSN user, to assess the quality of obtained data and detect errors or attacks. Even with a significant fraction of the WSN nodes controlled by the attacker, our scheme identifies and mitigates the effect of the attacks. This is supported by our analysis, with simulations of realistic strong attacks. The practicality of our scheme is supported by our proof of concept implementation.**

I. Introduction

Wireless sensor networks (WSNs) consist of tiny, low-power devices equipped with sensing, computation, and wireless communication capabilities. Each of those devices can monitor physical conditions, such as temperature, humidity, light, vibrations, etc. Networks composed of dozens of nodes are envisioned to open a variety of new technological perspectives. Data aggregation has been recognized as an important approach to enhance the efficiency in WSNs. Individual sensor node readings are used to compute a meaningful aggregate value that is of interest to the WSN user. The aggregation algorithm can be a simple formula, e.g., the average of sensor measurements in a certain area, or even a complex event detection that might involve different kinds of sensors. Data aggregation reduces the amount of data to be transmitted or stored, and it allows the WSN user to get only parameters of interest while hiding the detailed state of the system.

Often, the WSN will have to operate in an *unattended* manner: sensor nodes are deployed, and they collect and store measurements without the presence of a base-station or sink. Instead, the unattended WSN is self-organized, in terms of collecting multiple, over time, aggregate measurement values, storing them so that they can be later retrieved asynchronously by a mobile sink node (that is briefly present). Examples of such WSNs include systems for road condition monitoring or in situ habitat monitoring.

In general, reliable monitoring of parameters in such systems calls for a level of redundancy, e.g., with several nodes co-located in a small area. Each unattended WSN can be either a small network with a specific task or a cluster of nodes in a larger WSN responsible for a locally measured value. Naturally, nodes in such unattended WSNs are at least equally vulnerable to compromise or malicious attacks as those in any other WSN. Moreover, the absence of a trusted sink (that can be better protected than the sensor nodes) for most of the time, deprives the system from a facility to detect manipulation of the data collection.

The challenge lies in building a system that can mitigate misbehavior. We enable the mobile sink to assess if the aggregated data were manipulated even though it arrives at the unattended WSN long after the data collection and aggregation took place. The system should be resilient and deliver the *best possible* result in spite of the attack. Aggregation protocols that simply halt if an error is detected are not an option for our scenario.

We propose a new *resilient* data aggregation scheme that introduces a multi-dimensional *Quality of Information* (QoI) measure as integral part of the aggregation. The idea behind using QoI is that an attacker cannot manipulate the reported aggregate value from the WSN while successfully misleading the user the obtained QoI is high.

Our scheme is *flexible* to interoperate any application and aggregation function, and it is tailored to the requirements of unattended WSNs: The aggregation process is *autonomous* and it does not presume the presence of a sink. Aggregated data are stored in the WSN, for later collection by a mobile sink. Our analysis finds that, even with *half* of the measuring nodes affected or controlled by the adversary, our scheme will reveal the attack; or, even if the aggregating node is controlled by the attacker, any tangible manipulation of the collected aggregate will be detected. Overall, with increasingly many WSNs envisioned to operate in an unattended manner, our scheme can be a significant contribution towards their trustworthy and robust operation.

In the rest of the paper, we first discuss related work in the literature (Sec. II), and then we present in detail our system

978-1-61284-928-7/11/$26.00 ©2011 IEEE

and adversary models (Sec. III) and an overview of our scheme (Sec. IV). Details of our scheme components are provided next (Sec. V), before an analysis of the scheme resilience (Sec. VI), a discussion of practical issues through our implementation experience (Sec. VII) and our conclusion.

II. Related Work

Aggregation functions use redundant sensors to produce a reliable result, e.g., [17], [2]. We assume such an aggregation function to be present, and we additionally require that the aggregation function computes a precision measure for our scheme. However, such a resilient aggregation function alone is not sufficient: as the aggregation can be performed by non-trusted nodes (i.e., neither the sink nor the WSN user), a malicious aggregator can single-handedly control the aggregate.

A common solution to the problem of malicious aggregation is the "commit and attest" approach [18], [4]. First, the information is aggregated, without ensuring its correctness. Then, the base station initiates the validation of the aggregate. Variants of this approach [7], [15] achieve robustness at the expense of such additional sink-WSN interactions. Even though these schemes can evict attackers that manipulate the aggregation, they cannot be used in our setting: When an unattended WSN is queried by the user (e.g., a passing-by sink), its sensor nodes no longer have their prior own measurement contributed to the aggregation in question. Thus, they cannot participate in a subsequent manipulation detection round.

SecureDAV [10] can authenticate the result of an aggregation in an unattended WSN in one round, however, its use of public key cryptography makes it unsuitable for our more constrained environment. Moreover, it does not offer the robustness and QoI aspect as we seek. A witness-based approach is proposed in [5]: One aggregator node and m witness nodes are chosen. If n out of $m+1$ nodes authenticate the aggregated result, it is accepted by the base station. This provides robustness against node failures, but it does not offer any indication of QoI; especially if the condition about sufficiently many benign witness nodes is not met.

QoI for WSNs was studied in [8], which described how it can be measured and which parameters to chose. Several publications of application-specific quality evaluation methods have been proposed, with [19], [9], [11] considering water contamination, radar identification and databases respectively. A high-level computational framework for evaluating QoI in a detection-based sensor network was given in [20]. A fuzzy-based aggregation framework which provides in-network resilience was presented in [6]: A fuzzy controller measures and outputs a single QoI value. However, that single QoI value turns out to be limited in terms of protection it can offer against a range of attacks.

III. System Assumptions

A. System Model

An *unattended* WSN (*U-WSN*) consists of nodes with the same capabilities and roles changing dynamically over time. A U-WSN, a typically small network of few tens of nodes, can be stand-alone, or it can be part of a larger WSN, responsible for monitoring its locality. Nodes perform measurements at each "*epoch*," a system-specific period, which depends on factors such as the monitored phenomenon and the node deployment. We assume that U-WSN nodes form a connected cluster, i.e., every node has a direct link or a communication path to any other node. By design, the node are closely placed and their measurements are expected to be strongly correlated or even identical under ideal benign conditions. We do not dwell on the measured data model and the node deployment, as those would depend on the supported application.

Nodes execute a distributed protocol, propagating their measurements across the U-WSN, with an aggregate (e.g., average, min, max) over all contributed values calculated and stored locally. For reliability reasons, multiple nodes in the U-WSN (for simplicity, we refer to it interchangeably as the network in the rest of the paper) act as aggregators and store the aggregate values. The network user can retrieve aggregate data at any later point in time, e.g., with a mobile sink that interacts with the network only to have the aggregate result delivered. The sink is not required during the execution of the aggregation protocol, with the U-WSN operating autonomously.

The communication across the network can be done in different ways, e.g., by discovering neighbors (nodes directly reachable across the wireless medium) and by calculating communication paths to other nodes. Given the small size and small diameter of the network, each message can be flooded in a controlled manner (e.g., with a time-to-live). The exact communication protocol is orthogonal to the problem at hand. We assume that the network nodes are capable of such functionality (flooding, tree-based communication, unicast routing) using any of the schemes in the literature. We also assume that nodes are loosely synchronized: essentially, they are able to operate in the same epoch, i.e., reference the aggregation according to the time or date of computation. Time synchronization is not required for the security of our protocol. Its absence would lead to low QoI, and the level (loose/tight) of synchronization can affect the choice of authentication or node election protocols.

We assume that each sensor node has a unique identifier, *ID*, and shares a unique secret symmetric key, K_{ID}, with the sink. Each node also shares a symmetric key with each other node in the network, so that sensor nodes can exchange messages in an authenticated way. A simple method for the relatively small networks considered here is to use message authentication codes based on pair-wise secret keys. If a node falls under the control of the adversary, then it can impersonate only the node it compromises and it has no side information about the other keys. Alternatives with more flexibility are known in literature, e.g μtesla [12] or the multicast authentication by Canetti et al. [3].

A node election mechanism is needed for the assignment of roles to the nodes. Generally, an election process takes place at the beginning of an epoch. This can distribute the load over time, protect the resources, and it provides a certain

robustness as a failing node can be replaced in the next election. The node election process should be non-manipulable so that compromised nodes cannot have a higher chance of getting the vital role of aggregator or storage nodes. Various node election protocols for WSN are available in literature, e.g. [1], [14].

B. Adversary Model

The adversary can control a fraction, t_C, of the network nodes. Compromised nodes can act as *internal adversaries*, as they are equipped with the cryptographic keys that allow them to authenticate their messages to other nodes. Any internal adversarial node can forge and inject any message during the protocol operation (or simply withdraw from the protocol execution).

The adversary can control additional nodes acting as *external adversaries*, i.e., eavesdropping, replaying, and jamming transmissions of legitimate nodes. This can be performed by placing some devices that specifically act in that manner, or use the internal ones in that way, as jammers for example. For simplicity, we do not distinguish the two; we consider the stronger internal adversaries.

In our context, jamming, in particular *selective erasure of messages* is important. Without dwelling on the exact method of deliberate interference (at the physical or data link layer), and extending the notion to prevention of transmission of a message from node A to the aggregator node B, we assume that the adversary can jam *up to a fraction, t_J, of the node measurements* in an epoch. In other words, it can prevent the measurement of $100t_J\%$ of (legitimate) data contributed towards the aggregate calculation.

The goal of the adversary is to affect and possibly control the collected data from the network, notably the aggregate values. Here, we focus on the manipulation of a single aggregation, i.e., over one epoch, in the sense that we seek to counter misbehavior over each protocol execution (aggregation).

We do not consider outright denial of service, that is, any action that would prevent the U-WSN from receiving any data from one or more epochs. The reason is that this could be achieved in many ways that are orthogonal to the aggregation per se; for example, the attacker could disable any protocol by 'cutting' the network with a set of conveniently located jammers. Instead, we consider an attacker trying to mislead the user with seemingly correct data provided by the U-WSN, which in fact is manipulated and differs significantly from the actual measurements.

Based on the above, we primarily consider the following two main and most powerful relevant attacks:

- *False Data Injection:* each adversarial node contributes data that deviate from its actual measurement.
- *False Data Aggregation:* the aggregator node is adversarial; thus, independently of the node data contributions, the adversary can arbitrarily set the aggregate value.

and their combination with *selective erasure of messages* sent by benign (correct) nodes.

IV. Protocol Overview

Our resilient aggregation scheme is based on a two-step *aggregate-and-confirm* procedure, performed autonomously by the U-WSN. Within an epoch, each node assumes one out of three roles: (i) *normal*, (ii) *aggregator (AN)*, and (iii) *storage (SN)* node. Each node randomly picks one of the three roles at the beginning of each epoch.

Normal nodes need to first provide their measurement to the aggregator, which in turn collects all values, removes outliers and it computes an *aggregate*: the result *result*$_{agg}$ and a measure of precision *precision*$_{agg}$. The precision expresses the dispersion of the contributed, presumed "genuine" data set.

Then, the aggregator returns the aggregate to the (normal) nodes. In turn, each receiving node controls if the aggregate is correct: It compares its own measurement to the aggregate, i.e., checking if it falls within the precision interval around the result. If so, the normal nodes calculate a *confirmation*: a digest using the symmetric key they share with the U-WSN user. We describe the requirements and properties of the aggregation function and its respective confirmation function in Section V.

The confirmations are collected by the storage nodes, which keep those and the aggregate (as issued by the aggregator). They store those until requested by the U-WSN (or purged after a system-selectable time-to-store period elapses).

This three-phase exchange is illustrated in Fig. 1. The messages, with HDR being the protocol header, are:

1 Data message [HDR; $measurement$]
2 Aggregate message [HDR; $result_{agg}$; $precision_{agg}$]
3 Confirmation message [HDR; $confirmation\ digest$]

For each epoch, one node is elected as the aggregator, and one or multiple (a protocol-selectable number of) nodes as storage nodes. The remaining majority are normal nodes; clearly, the storage and the aggregator nodes also act as normal nodes: the storage ones send their measurement and the aggregator adds its own to its aggregate computation.

The communication overhead of the three-phase exchange can be kept low. The aggregator initiates an authenticated broadcast, which serves to establish a tree rooted at the aggregator. The same authenticated message serves as a trigger for normal nodes to respond after a delay (much shorter than the epoch) with their measurements.

Each receiving (triggered) normal node randomly elects if it will act as a storage node. If so, when it receives the aggregate, it retains it. Then, the confirmation messages from all nodes are sent towards the storage nodes (SNs). The identity (and the placement in the aggregation tree) of each storage node can be piggy-backed in its measurements with an additional broadcast authenticated field in its header.

The base station does not play any role during the aggregation, it only later retrieves the aggregate for the user. For example, an authenticated broadcast requesting aggregates from one or more epoch can be issued. The SNs from that epoch shall respond with the aggregate and the list of confirmation messages. Thanks to this information, the U-WSN user will be able to compute a measure of the *Quality of Information* (QoI).

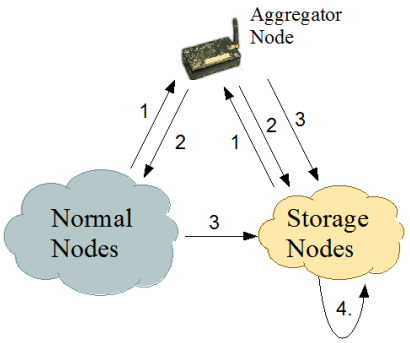

Fig. 1. Protocol overview: 1. Network sends measurements 2. Aggregation of data at the AN; sending back aggregate $[result_{agg}; precision_{agg}]$ 3. Nodes check the result and send a confirmation message to the SN.

The choice of this QoI measure is such that, as explained next, it mitigates the manipulation of data. Essentially, it constraints the adversary, even if powerful, in terms of how much it can distort the view of the data the U-WSN gets. At the same time, the user has a clear indication of the quality of data it obtains (possibly severely manipulated, or not yet unreliable, etc), and it can act/use them accordingly.

V. Protocol Components

Aggregation Function: This is a central block of the scheme, taking as input the readings $x_1, \ldots, x_n$ of the n sensors in the U-WSN. It outputs a pair of values: the aggregation result, $result_{agg}$, and a measure of the data dispersion, $precision_{agg}$:

$$Agg(x_1, \ldots, x_n) = result_{agg}; precision_{agg}$$

The precision could be related to the standard deviation or a min-max interval. The goal of this parameter is to provide normal nodes with an indication of the spread of the data, potentially contributed by the correct nodes.

The performance of the protocol depends on this function and how outliers are removed from the set. Indeed, for the framework to be resilient, the aggregation function must be resilient, i.e., the data aggregation function must remove or limit the effect of outliers and tolerate missing values. Constructions of resilient aggregation functions are provided in [17], [2].

Confirmation function: This is coupled with the aggregation function to check the correctness of the aggregated result.

$$Conf(result_{agg}, precision_{agg}, x_i) = \texttt{decision}$$

The confirmation function takes $(result_{agg}, precision_{agg})$ and the own measurement, x_i of node i, as inputs; it outputs a decision, that is, one of $\{\texttt{ACK}, \texttt{NACK}\}$. The aggregation function is computed only by the aggregator node, while the confirmation function by every node.

A confirmation, $conf_i$, is used by nodes as a medium to communicate the decision about the result to the base station. Each node can agree on the result (ACK) if it is plausible according to the node's own measurement, or disagree (NACK) otherwise. The decision is wrapped inside a confirmation digest, encrypted with a symmetric pairwise key $K_{n_i,BS}$, shared by the node i and the base station. The digest consists of a truncated MAC of the tuple $(result_{agg}, precision_{agg})$ and the agreement decision. The length depends on the *per-confirmation unforgeability* we want to guarantee. In this case, we set the length to 8 bits:

$$conf_i = \text{MAC}^{8-bits}_{K_{n_i,BS}}(\texttt{ timestamp} \parallel result_{agg} \parallel precision_{agg} \parallel \texttt{decision})$$

A. Quality of Information

Simply put, *Quality of Information* (QoI) is an indication or a sort of feedback on whether the data resulting from an aggregation are faulty. This feedback can be elaborate, in the sense it indicates how much can the user trust the result, how accurate or how recent it is. All such information can be very beneficial.

As confidence in or accuracy of aggregate data are not easy to measure, we define new QoI characteristics that are easier to measure and they are uncorrelated: *Precision*, *Network Reliability*, and *Aggregator Reliability*, and *Freshness*.

- *Precision:* It indicates if the measured values of different sensors are close to each other. Precision is used as a proxy accuracy, i.e., the proximity of the aggregate returned by the protocol to the actual value (i.e., the aggregate value of all actual measurements). But determining the actual value in the presence of faulty (adversarial) sensor measurements. This is why we use the *Precision* parameter.
- *Aggregator Reliability:* It indicates the quality of the aggregation, and it is computed as a ratio: the number of nodes that confirmed (ACK-ed) the aggregate, over the number of nodes that sent any *confirmation messages* (ACK or NACK). The quality of the aggregation result, the ratio, approaches one as the number of ACK-ing nodes increases. More NACK-ing nodes imply the quality decreases, going to 0, indicating it is likely the aggregator did not compute properly the result.
- *Network Reliability* captures the network state: the number of nodes that confirmed (ACK-ed or NACK-ed), over the total number of the U-WSN nodes present. In other words, this ratio captures the "general health" of the network, distinguishing responding, i.e., alive and able to communicate, nodes from the rest (off-line, cut-off, or jammed).

  The two reliability measures are calculated based on the retrieved `confirmation messages`:

$$\text{Aggregator Reliability} = \frac{\text{N}_{ACK}}{\text{N}_{ACK} + \text{N}_{NACK}}$$

$$\text{Network Reliability} = \frac{\text{N}_{ACK} + \text{N}_{NACK}}{\text{N}_{tot}}$$

  where N_{ACK} is the number of nodes that agreed with the result, N_{NACK} the number of nodes that disagreed with the result and N_{tot} is the total number of nodes present in the network (including nodes that did not contribute at all to the result of the epoch).

- *Freshness:* It determines if the aggregation process remains still valid (recent) or expired (outdated). It is user-specific.

  $$\text{Freshness} = \min(\ \max(\ \frac{\Delta timestamp - \tau_{min}}{\tau_{max} - \tau_{min}}, 0), 1)$$

  where $\Delta timestamp$ is the difference between the current time-stamp and the one at the time of aggregation (at the granularity of an epoch, at least):

  $$\Delta timestamp = timestamp_{current} - timestamp_{aggregation}$$

  τ_{min} and τ_{max} are time boundaries for deeming data fresh (or expired), $\Delta timestamp$, τ_{min} and τ_{max} can be measured as the number of epochs/aggregations.
  Freshness is trivially achieved as all nodes ensure they operate and perform an aggregation within the same epoch, and when the user (passing by sink) determines which aggregate it seeks to retrieve. In what follows, we focus in the first three QoI parameters: *Precision*, *Aggregator Reliability*, and *Network Reliability*.

VI. Scheme Evaluation

A. Security Analysis

The integrity of in-transit messages (data, result, and confirmation) is achieved thanks to authentication (based on pairwise symmetric keys), and adversarial nodes cannot forge and transmit messages on behalf of other nodes. Replayed messages across or within the same epoch are also detected, trivially, across different epochs; within the same epoch, the first authentic contribution by any node is considered. The attacker can basically *inject false data*, *falsely aggregate data*, and selectively erase messages, as explained in Sec.III-B.

We explain first qualitatively how our design prevents an adversary from mounting an effective attack. Then, we present experimental (simulation) results to quantify the attack effect and the resistance of our scheme. Basically, it is hard for the adversary, Adv to distort the aggregation result and at the same obtain high *Precision (Prec)*, high *Aggregator Reliability (Rel A)* and high *Network Reliability (Rel N)*. Consider the effects on the QoI for Type *I* (*false data injection attack*) and Type *II* (*false data aggregation attack*) adversaries:

High Precision

- To cause high distortion while getting a falsely high *Precision*, Adv needs to compromise sufficiently many nodes so that the false values will be in the majority and the aggregation function will ideally discard the genuine values as outsiders. By doing so, Adv achieves high *Precision*, yet the resultant *Aggregator Reliability* will be low: the remaining benign nodes will not confirm the result if the distortion they perceive is high (NACK).
- Given Adv can control the *Precision*, it then needs to chose carefully the level of distortion it wishes to cause: Essentially, it needs to avoid low *Aggregator Reliability*, i.e., keep distortion low in order to collect ACK from benign nodes. But this exactly implies the effect of the attack, admittedly perpetrated, would be low.

High Aggregator Reliability

- Alternatively, Adv can jam benign nodes, erasing their NACK messages, in order to keep *Aggregator Reliability* high (in the above setting, i.e. while causing high distortion and high precision). But doing so will bring down the *Network Reliability* metric, as less nodes will send confirmation messages.
- In other words, Adv needs many nodes to acknowledge its aggregate. It could cause high distortion but send a low *Precision*. The Adv sacrifices *Precision* for *Aggregator Reliability*. Jamming of confirmation messages would not be of much help as nodes are expected to confirm that the result is a low precision value.

B. Simulation Results

1) Scenario: We simulate a small sensor network, with sensor measurements drawn pseudo-randomly for each node and each iteration, with the following parameters:

- *Network size*: 20 nodes
- *Distribution of the data:* $N(\mu = 100; \sigma = 2)$
- *Number of iterations:* 1000

We evaluate both the *resilient average* and the *resilient maximum* aggregation functions, which operate as follows: (i) outliers are removed from the sensor measurement set, (ii) the remaining measurements are input to basic *max* or *average* operators. As defined in Sec. V, the aggregation function must compute a precision and offer a confirmation algorithm. Below is one example of precision and confirmation algorithms (in `python` code); simulations were written in `python` and run on `sagemath` [13].
Average function:

```
def function_avg(data):
    avg = 0.0
    for d in data:
        avg += float(d) / len(data)
    //difference between the value the furthest
    //away from the average and the average:
    precision = max( abs(max(data) - avg),
                     abs(avg - min(data)) )
    return [avg,precision]

def confirmation_avg(item,avg,precision):
    if (avg - precision <= item  <= avg + precision)
        return 1 //acknowledge
    else:
        return 0 //does not acknowledge
```

2) Adversary: We simulate a full-blown adversary, i.e., (i) *False data injection*, (ii) *False data aggregation*, and (iii) *jamming*. The difference between the reported (adversary-affected) result and the actual value is termed *distortion*. To relate the distortion to the actual value, we measure the *distortion* values chosen by the adversaries as multiples of the variance of the measurements' distribution, σ.

Type I Adversary uses both *false data injection* and *jamming*. We vary the following parameters:

- Number of nodes influenced: Adv can modify the sensor measurement of 0 to 10 nodes, i.e. has 0% to 50% of the WSN under control.

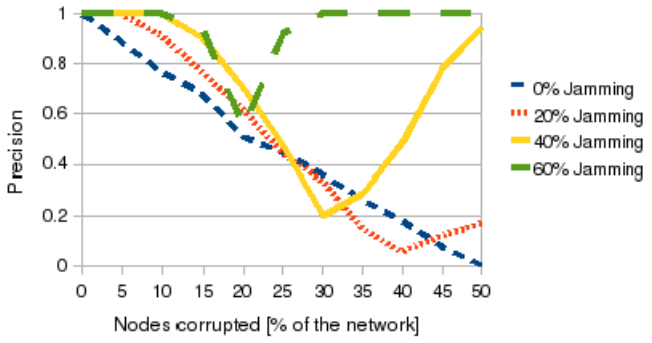

Fig. 2. Precision with a Type I attacker (attacking maximum aggregation with a false data injection attack). Four levels of additional jamming of honest nodes are considered.

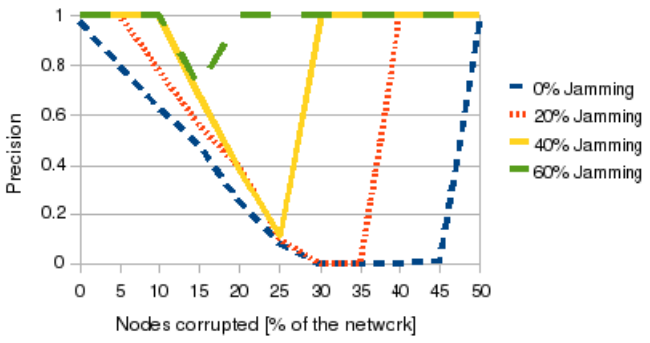

Fig. 3. Precision with a Type I attacker (attacking maximum aggregation with a false data injection attack). Four levels of additional jamming of honest nodes are considered.

- Amount of distortion injected: Adv increases at all nodes it controls their actual measurements by a *distortion* value in $[\sigma, 5\sigma]$. Larger distortions do not provide new insights, the trend within this interval turned out to be clearly visible. For an average aggregate function it simply does that. For a maximum aggregate, Adv increases its actual maximum measurement, *max*, by a chosen *distortion* value.
- Number of nodes jammed: Adv removes messages of up to 60% of the nodes.

Type II Adversary controls the aggregator node (*false data aggregation*) and it can also launch a *jamming attack*. We vary:

- Amount of distortion injected: Adv adds a chosen *distortion* value to the actual, correct aggregation value.
- Aggregate precision value: Adv can freely chose (e.g., decrease) the precision, e.g., to increase the number of nodes that confirm the aggregation result.
- Number of nodes jammed: Adv removes messages of up to 60% of the nodes.

The adversary 'wins' when the aggregated value is altered by the sought distortion and the erroneously perceived QoI is high.

3) Results for Type I attack:

a) Precision: We normalize the precision given by the aggregation function, i.e., in $[0, 1]$:

$$\text{Precision} = min(max(\ 1 - \frac{\text{pres}_{agg} - \varepsilon}{\delta - \varepsilon}\ ;\ 0)\ ;1)$$

$pres_{agg}$ is the precision of the aggregate, what the aggregator node outputs. The parameters ε and δ are used for normalization and have to be determined from the properties of the measured phenomenon.

For our scenario, $\varepsilon = 3.16$ and $\delta = 4.74$ are plausible for the average case and $\varepsilon = 2.86$ and $\delta = 4.29$ for the maximum aggregation. These values came from our experiments (measuring temperature, please see our proof-of-concept implementation description, in Sec.VII), and they are application- and monitored phenomenon- specific.

Fig. 2 shows how *Precision* is affected under jamming and false data injection attacks (for the average as the aggregate function). It first remains at 1, then drops linearly and at a certain point in time it changes behavior and starts increasing linearly with t_C. The lowest point corresponds to a configuration where the adversary starts winning, achieving high distortion. Indeed, the aggregate result is modified as soon as the aggregation function cannot consider the wrong values to be single outliers anymore. *Precision* increases again at the point where the aggregation function considers the genuine measurements to be outliers. Note that the distortion does affect this experiment: in order to have only a σ distortion on the average, the attacker has to increase the measurements at w nodes by the value $\frac{n \times \sigma}{w}$. Such values, highly uncorrelated to the genuine data set (even for a small distortion), leave as the only possibility for the adversary to compromise at least half of the nodes. This is the reason why we grouped the distortion levels together. Therefore, for the attack to succeed, half of the remaining nodes (or more) have to be compromised.

Jamming, however, plays a very important role in this attack, in fact, it can be used to support the false data injection attack. Fig. 3 shows how *precision* varies for three levels of distortion. Curves behave similarly for both aggregate functions (average, maximum). First, jamming helps the adversary that injects false data: the more legitimate messages it can erase, the fewer nodes it needs to compromise. Second, for a few compromised nodes, curves for high distortion or high jamming remain flat and close to 1. This is because, in a preprocessing, the aggregation function can easily detect and remove outliers.

b) Aggregator Reliability: We compute this with the following formula:

$$\text{Aggregator Reliability} = \frac{\text{N}_{ACK}}{\text{N}_{ACK} + \text{N}_{NACK}}$$

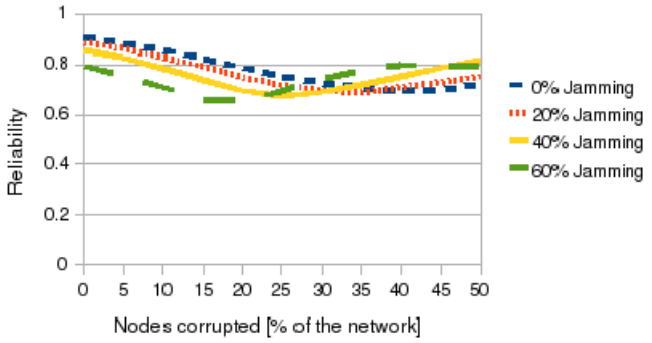

Fig. 4. Aggregator reliability with a Type I attacker (attacking average aggregation with a false data injection attack). Four levels of additional jamming of honest nodes are considered.

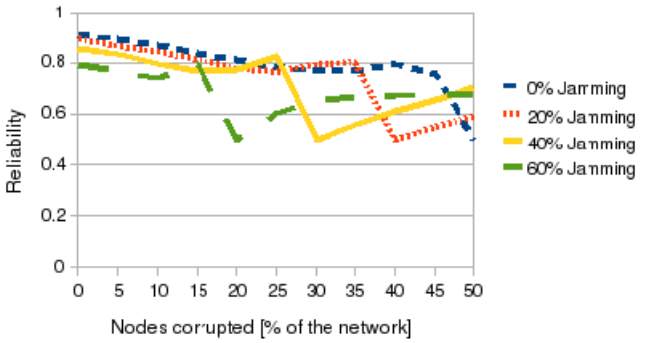

Fig. 5. Aggregator reliability with a type I attacker (attacking average aggregation with a false data injection attack). Four levels of additional jamming of honest nodes are considered.

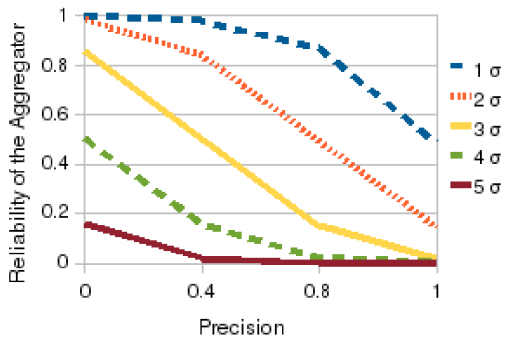

Fig. 6. Trade-off between aggregator reliability and precision achievable by a malicious aggregation node (Type II adversary); average aggregation.

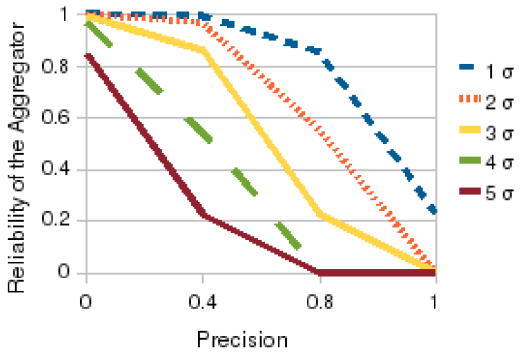

Fig. 7. Trade-off between aggregator reliability and precision achievable by a malicious aggregation node (Type II adversary); maximum aggregation.

Fig. 4 shows that *Aggregator Reliability* remains relatively high and obtains a minimum when the attack succeeds. Initially, it decreases with t_C and then it goes up again. It quantifies the percentage of nodes confirming the result, therefore the minimum is reached when half of the nodes (correct or compromised) ACK and the other half NACK.

With maximum as the aggregate function, Fig. 5 shows clearly when the adversary succeeds. For small and medium distortions, the curves have a particular behavior: they are initially almost flat, increasing slightly and at the *winning* point they drop to 0.5 (= 50% of nodes acknowledged). Then, the *Aggregator Reliability* increases a little bit and stabilizes. As it is computed from the number of ACKs, it depends on the aggregate precision: the lower (wider), the more nodes would ACK. This also depends on the pre-processing function: removing outliers shrinks the max-min interval (i.e., increase the precision). The sudden slope represents the exact moment when the aggregation function starts dropping the genuine values as outliers and keeping the forged ones, the precision interval is so narrow that no other nodes than the compromised ones can confirm. Final remark: the higher the distortion, the more similar are the behavior for the maximum and the average function. This was expectable, as both functions (average and maximum with high distortion), inject values much higher than the original ones.

c) Reliability of the Network: The Network Reliability is calculated as:

$$\textit{Network Reliability} = \frac{\mathrm{N}_{ACK} + \mathrm{N}_{NACK}}{\mathrm{N}_{tot}}$$

and it is the ratio of nodes that did not send a ACK or NACK message.

The Network Reliability depends only on the Jamming ratio and it is totally independent to the number of corrupted nodes and the (resilient) aggregate function.

C. Results for Type II Attacks

The adversary, having compromised the aggregator node, can provide a (highly) distorted value and a (forged) aggregated precision. Unlike the Type I adversary, a Type II adversary always succeeds in modifying the final aggregate value within the epoch it happens to act as an aggregator. However, the quality of information would vary with the incurred distortion and the aggregate precision broadcasted to the network. The attacker should mislead the querying user that the QoI is high in order to *win*.

Recall the QoI composed of the following parameters:

- ***Precision:*** Depends on the computed value of the aggregate precision.
- ***Network Reliability:*** As for Type I adversaries, it depends on the jamming ratio.
- ***Aggregator Reliability:*** Depends on the number of nodes who confirmed the aggregated result, taking into account the aggregate precision.

For Type II attacks, jamming does not help because the parameters the adversary wants to influence (or even maximize in order to maximize the effect of its attack) are orthogonal to jamming. Therefore, we do not show here plots of the Network Reliability.

We exhibit the trade-off Type II adversaries face: *Precision* vs. *Aggregator Reliability* vs. Distortion injected. The adversary has an *advantage* when the curve is as close as possible to the external (1;1) point (*win*). On the contrary, if the curve is close to the origin (0;0), the adversary *loss*.

Fig. 6 illustrates the QoI-trade-off for aggregation of an average value. As we expected, for high distortion the Aggregator Reliability is low (even with low Precision). But for small distortions (σ - 2σ), the attacker can still achieve acceptable Reliability for a relatively high Precision. The adversary cannot have both Precision and Reliability.

Fig. 7 illustrates the same trade-off for the maximum as aggregate function. The main difference from Fig. 6: high distortions and low precision now have high Reliability. More generally, the *max* function is less robust to the Type II adversaries than the *average* function. This lies in the nature of the max function, which is looking for an extreme value, even in it's resilient variant.

VII. Implementation

We provide a proof-of-concept implementation that shows the feasibility and practicality of the scheme. The emphasis and contribution of this paper is the investigation of the security problem and the achieved resilience. Fine-tuning the implementation, on a per case study basis, can be based on the findings of the community and it is interesting, but a secondary objective for this work. We leave this and a detailed performance evaluation (message complexity, delays, communication reliability, etc) as part of future work.

We implemented our scheme and run it in a testbed of *TelosB* and *Tmote Sky* sensors. The operating system of those motes is called TinyOS: it is open-source and has been designed for wireless embedded sensor networks. TinyOS has been ported on many platforms and currently over 500 research groups and companies are using TinyOS [16].

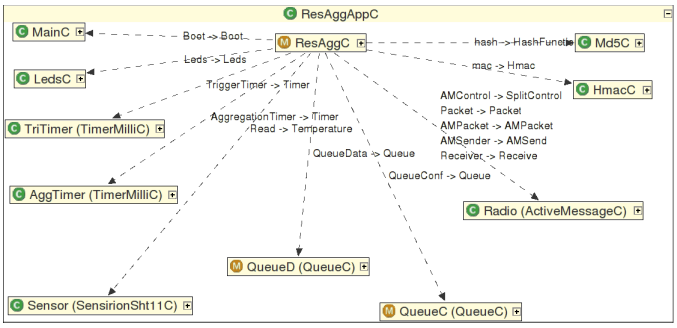

Fig. 8. Layout of our implemented modules, components and interfaces.

The TinyOS system, libraries, and applications are written in nesC, which is a language for programming structured component-based applications and has a C-like syntax. nesC application consists of one or several **components** which provide and use *interfaces*. Interfaces provide a set of functions and/or events that the interface provider has to implement. There are two different types of components: the **modules** and the **configurations**. Basically, modules provide the code for the application and configuration link the components together: it **wires** (connect) the interfaces to the components. The two main components are *ResAggC* (module) and *ResAggAppC* (configuration file): *ResAggC* uses interfaces of scheme components and implements their functionality with *ResAggAppC* the configuration file that instantiates and wires those interfaces to the module; Fig. 8 shows a diagram of *ResAggC* components. The module implements ten interfaces. *MainC* is the principal one, which does the link between the application and the OS. There are two timers *TimerMilliC* used by the aggregator node to rhythm when to trigger and aggregate. The *SensirionSht11C* controls the sensor elements: it can read the temperature and the humidity. Note that in our implementation, we decided to measure the temperature because it is easier to "visualize". Another important interface is *ActiveMessageC*, which manipulates the radio controls: send and receive messages. For the confirmation messages and the authentication broadcast, *Md5C* and *HMAC* are used.

VIII. Conclusions

We presented a novel approach for data aggregation which provides resilience in an *unattended* WSN. Resilience includes robustness to attacks and node disappearance, integrity of the measurements, and providing a Quality of Information (QoI) measurement. This QoI is a key concept to mitigate failures and malicious attacks, as it is not possible to succeed in both, convincing the user of a inaccurate value and providing a high QoI value.

Our system autonomously aggregates and stores the result within the WSN, not requiring a sink node during operation. When a mobile sink node is temporarily available, it can query the WSN to retrieve present or past aggregation results.

The evaluation was done by extensive simulation of a medium-sized scenario. Various attacks with malicious nodes and jamming have been simulated and results presented. A proof-of-concept implementation was done in tinyOS.

A further evaluation is left for future work. Some points to address are the evaluation of the protocols in a network of larger scale and a detailed evaluation of the protocol overhead.

ACKNOWLEDGMENT

The work of the authors from NEC was partly funded by the European Commission within the STREP WSAN4CIP project. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsement of the WSN4CIP project or the European Commission.

REFERENCES

[1] L. Buttyán and P. Schaffer, "Panel: Position-based aggregator node election in wireless sensor networks," *International Journal of Distributed Sensor Networks*, September 2008, (submitted).

[2] L. Buttyán, P. Schaffer, and I. Vajda, "Ranbar: Ransac-based resilient aggregation in sensor networks." in *Workshop on Security of Ad Hoc and Sensor Networks, SASN 2006*, 2006, pp. 83–90.

[3] R. Canetti, J. A. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas, "Multicast security: A taxonomy and some efficient constructions," in *INFOCOM '99*. IEEE, 1999, pp. 708–716.

[4] H. Chan, A. Perrig, and D. Song, "Secure hierarchical in-network aggregation in sensor networks," in *ACM conference on Computer and communications security, CCS '06*. ACM Press, 2006, pp. 278–287.

[5] W. Du, J. Deng, Y. S. Han, and P. Varshney, "A witness-based approach for data fusion assurance in wireless sensor networks," in *IEEE Global Communications Conference, GLOBECOM 2003*. IEEE, 2003.

[6] E. De Cristofaro and J.-M. Bohli and D. Westhoff, "FAIR: Fuzzy-based Aggregation providing In-network Resilience for real-time Wireless Sensor Networks," in *Proceedings of the 2nd Symposium on Wireless Network Security (WiSec'09), ACM*, 2009.

[7] P. Haghani, P. Papadimitratos, M. Poturalski, K. Aberer, and J.-P. Hubaux, "Efficient and Robust Secure Aggregation for Sensor Networks," in *Proceedings of the Third IEEE ICNP Workshop on Secure Network Protocols (NPSec)*, Beijing, China, October 2007, pp. 1–6.

[8] H. Karl, "Quality of service in wireless sensor networks: Mechanisms for a new concept?" in *European Science Foundation (ESF) Workshop, Zurich, Switzerland*, April 2004.

[9] J. Kumar, E. M. Zechman, and S. R. Ranjithan, "Evaluation of Non-Uniqueness in Contaminant Source Characterization Based on Sensors with Event Detection Methods," in *In Proceedings of the World Environmental and Water Resources Congress 2007*, 2007.

[10] A. Mahimkar and T. S. Rappaport, "Securedav: A secure data aggregation and verification protocol for sensor networks," in *IEEE Global Telecommunications Conference, GLOBECOM '04*. IEEE, 2004.

[11] F. Naumann, U. Leser, and J. C. Freytag, "Quality-driven integration of heterogeneous information systems," in *In VLDB Conference*, 1999, pp. 447–458.

[12] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler, "SPINS: Security Protocols for Sensor Networks," *Wireless Networks*, vol. 8, no. 5, pp. 521–534, 2002.

[13] Sagemath, "Sage, a free open-source mathematics software system licensed under the gpl," 2009, http://www.sagemath.org.

[14] M. Sirivianos, D. Westhoff, F. Armknecht, and J. Girao, "Non-manipulable aggregator node election protocols for wireless sensor networks," in *In Proceedings of the International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, 2007.

[15] G. Taban and V. D. Gligor, "Efficient handling of adversary attacks in aggregation applications," in *Computer Security - ESORICS 2008, 13th European Symposium on Research in Computer Security, Málaga, Spain, October 6-8, 2008. Proceedings*. Springer, 2008, pp. 66–81.

[16] TinyOS, "Open-source operating system designed for wireless embedded sensor networks," 2009, http://www.tinyos.net.

[17] D. Wagner, "Resilient aggregation in sensor networks," in *Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks, SASN '04*. ACM Press, 2004, pp. 78–87.

[18] Y. Yang, X. Wang, S. Zhu, and G. Cao, "Sdap: A secure hop-by-hop data aggregation protocol for sensor networks," in *ACM international symposium on Mobile ad hoc networking and computing, MobiHoc '06*. ACM Press, 2006, pp. 356–367.

[19] B. Yu and K. Sycara, "Learning the quality of sensor data in distributed decision fusion," in *In Proceedings of the Ninth International Conference on Information Fusion*, 2006.

[20] S. Zahedi and C. Bisdikian, "A framework for qoi-inspired analysis for sensor network deployment planning," in *Proceedings of International Workshop on Performance Control in Wireless Sensor Networks (PWSN)*, October 2007.